VU Research Portal

Notes on Design Reasoning Techniques

Tang, A.; Lago, P.

2010

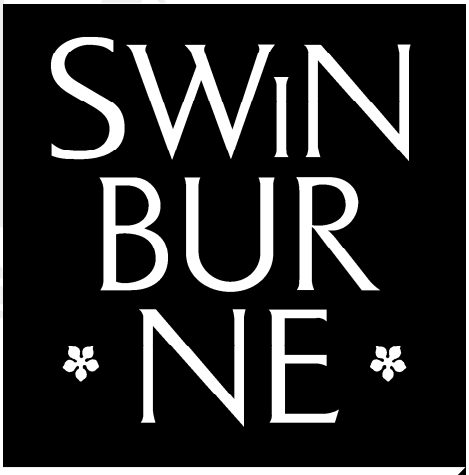
Technical Report

Notes on Design Reasoning Techniques

SUTICT-TR2010.01

Antony Tang[1], Patricia Lago[2]
[1]Swinburne University of Technology
[2]VU University Amsterdam

18th Aug 2010



SWINBURNE UNIVERSITY
OF TECHNOLOGY

Table of Contents

Notes on Design Reasoning Techniques

1. Introduction

The ability of a designer to reason effectively is an important skill in software design. We postulate that there are three key skills that a designer must possess to make good design decisions. They are knowledge, reasoning and creativity. The knowledge that a student or a designer of software must possess can be trained. That knowledge comprises of knowing the characteristics of software systems and products, e.g. how does JavaScript work; and of knowing some design methodologies, e.g. object-oriented design and analysis.

However, given a real-life design situation, a designer must possess the skill to recognise issues, and find ways to design a solution to solve the issues. Design reasoning techniques attempt to fill the gaps in this area, which is often not taught explicitly. Creativity is a mental process involving the creation of new concepts. It has been studied widely in the field of psychology but in the software industry, we know little about this to train a person to be creative.

In this document, we do not discuss the software engineering techniques that are used in software design, nor do we discuss how to be creative in design. Instead, we discuss several reasoning techniques that can be used to help a designer reason with his/her design.

2. Recognising and Formulating a Design

In order to design something, we intuitively formulate a design issue(s) or identifying they key design issues before proceeding to solve that issue(s). Two steps are useful in design issue identification: Firstly recognising what design issues need to be addressed and, secondly recognising what design concerns are driving a design issue.

If we do not know that a design issue exists, we basically will not design for it and there would be a gap in the solution. A good starting point is to consider the requirements of a system, using them to find the design issues. However, this is not enough as requirements are seldom specified completely and with enough details to drive a design to completion, additionally there are many other forces that influence a design. A design issue arises from a combination of requirements and other factors, technical and non-technical. If we do not *relate all* the relevant design concerns when considering a design decision, we may either miss an important design issue, or we may think a solution is workable when it is actually not.

So we need to relate design concerns in order to identify design issues, and also uncover design issues that need to be addressed by referencing design concerns. It seems like a “chicken and egg” situation. However, good designers approach these gaps from both angles until all related design concerns are identified and all design problems are addressed,

A key phenomenon in designing a software system is creating something quite unique given a complex set of situations and requirements. The set of requirements and situations may not be entirely explicit and then they would change as a design starts to take shape. One key aspect of the design process is to recognise the existence of the unknowns in a design, and to uncover all these

unknown design issues that need to be dealt with in a design. If design issues are not uncovered, we will not deal with them and they will affect the quality of the end system. To address the issue of "*We don't know what we don't know*", and to ensure that we make sound design decisions, we propose a number of design reasoning techniques.

3. Design Reasoning Techniques

As discussed earlier, a key step in design is to find out what design issues need to be dealt with. A designer could start off with problem structuring, It means to find out what is problems are in the design. Designers can also use constraint analysis, deductive and inductive reasoning to identify some of the design issues that need to be dealt with. The formulation of design issues inevitably lead to considerations of options and solutions. In considering the solution space, designers can use option analysis, risk analysis, constraint analysis and trade-off analysis to reason with the choice of design solutions. A number of design reasoning techniques are discussed in this section and they can help us think through a design. The following is a brief description of these reasoning techniques.

3.1. Problem Structuring

When designing something, we sometime deal with situations that we have not encountered before. How do we know what to do? We have to be aware of how to approach the design problem, understand what design problems are to be resolved. Ask yourself these questions throughout the design:

- What are the key issues of the design?
- Do we have enough knowledge to achieve the goals of the system?
- Do we understand all the requirements? Are the requirements complete?
- Are the quality attributes of the system completely specified?
- Can the software and hardware components that we choose satisfy the quality and functional requirements?
- Are there any aspects of the design that we have missed?
- Are there any gaps in our understanding of the problem and the solution?
- Are there any relevant context that would influence our understanding of the problem?

These questions should be asked repeatedly from when we start a design to when we are fully satisfied that our design would work. It means that we should continue to identify any potential gaps in our design until we are satisfied that we have covered all aspects of the design. By practicing this tactic, we aim to minimise the error of overlooking key issues in design.

3.2. Constraint Analysis

Constraint analysis is a technique to help assess if a design is viable or not. In order to assess the impacts of constraint(s) on a design decision, a designer must first recognise the existence of constraints. It means that the designer must recognise what design concerns would influence a design decision. A designer must search through the problem space and identify any potential constraints that may influence a decision. When designing a system, Ask yourself these questions when you consider each design decisions:

- What are the technical and requirement constraints in implementing the design?

- Do you have the skills to design and program this thing?
- Will the design satisfy the performance requirements?
- Is security an issue? Is the design secure?
- Are there any quality attributes that should be considered? How will they constrain your design?

Constraints can come from different sources such as functional requirements, non-functional requirements and project context. Your earlier design decisions also constrain on what you do next. So it is useful to recognise that certain constraints have a large scope of impacts, for instance, a tight project schedule will impact on all parts of your design, i.e. you may have to simplify your design to fit the project schedule.

Constraint can propagate through a series of decisions. For instance, a constraint on the physical size of a piece of computing equipment propagates through to all the components within that equipment.

Constraints can metamorphize or change in nature. If there is a constraint on the size of battery, then a decision may be to find a light-weight battery, but a light-weight battery has limited power which creates a new constraint on the power-usage of a device we may use.

In a series of inter-related design decisions, designers have to ensure that the constraints in each decision are satisfied and when put together they are consistent and create no design conflicts. When you see constraints that are conflicting, e.g. we cannot have a light-weight battery and enough power to drive a device, a design conflict has arisen and some compromises would need to be made. It means that certain constraint will need to be relaxed in order to yield a solution. In this case, either have a shorter battery life or design to use a heavier battery [1, 2].

3.3. Options Analysis

Researchers in psychology have proposed that there are two distinct cognitive systems underlying reasoning. System 1 comprises a set of autonomous subsystems that react to situations automatically, they enable us to make quicker decisions with a lesser load on our cognitive reasoning. System 1 thinking can introduce belief-biased decisions based on intuitions from past experiences, these decisions require little reflection. System 2 is a logical system that employs abstract reasoning and hypothetical thinking, such a system requires longer decision time and it requires searching through memories. System 2 permits hypothetical and deductive thinking [3, 4]. Under this dual process theory, designers are said to use both systems.

It seems that, however, designers rely heavily on prior beliefs and intuition rather than logical reasoning, creating non-rational decisions. Furthermore, the comprehension of an issue also dictates how people make decisions and rational choices [5]. The comprehension of design issues depends on, at least partly, how designers frame or structure the design problems. In other words, the dual process theory says that designers use prior beliefs or intuitions as well as reasoning. However, it seems that designers rely heavily on prior beliefs and intuition rather than a logical reasoning process, causing designer's "rational thinking failure" [3]. These

findings are also confirmed in a more recent study on decision making in software design where designers make use of rational and naturalistic decision making tactics [6].

One technique to overcome such issue is to use option analysis. At each decision point, a designer must ask what options are available to solve the problem. This activity requires a designer to search through the knowledge base, externally and mentally, to identify relevant knowledge to help create the design solutions. It also requires a designer to creatively compose possible design options. We do not know enough to teach how that creativity can come about. However, we have found that designers who are able to consider multiple design options at each decision point come up with a better quality design [7]. We have also noticed that it is habitual, especially for inexperienced designers, to consider the first design option that comes into mind as the final solution. Experienced designers seem to reason with design options differently in that they consider more design options and eliminate those that do not work well. You can ask yourself these questions when you consider each design decision:

- What options are available to me to address a decision issue?
- Can I relax some requirements or constraints so that I have more design options?
- Will a decision that I make now limit my future options?

3.4. Trade-off Analysis

Trade-off analysis is a technique to help assess and make compromises. There are different trade-off analysis techniques. A very simple idea is to use weighted-additive model to calculate the expected return of a design. For instance, a higher weight is given to a requirement to compensate for a compromise of a lower weight requirement [8]. In other words, this technique helps to prioritise what is important and with priority over what is unimportant.

Architecture Trade-off Analysis Method (ATAM) [9, 10] is a method to assess the utility of the design. It is a weighting or priority given to different requirements as they are decomposed into sub-requirements.

You can also argue qualitatively the pros and cons of a design. The reasoning should consider the costs and benefits of a design, the weaknesses and strengths of a design, the risks and non-risks of a design. In such argumentation, you need to identify the priorities and the reason why such priorities should exist [11, 12]. Trade-off analysis can be applied to all key decisions in a design. You should ensure that you have assessed the relative pros and cons of each design option. Ask yourself these questions when you encounter conflicting requirements or design:

- Is there a situation where you find that your design cannot satisfy both requirements at the same time?
- Is there a situation where you find that your design is not viable because there are conflicts between them?
- What are the pros and cons of each option that you consider?
- How do you make compromises?
- Why do you choose one option over the other ones?

3.5. Risk Analysis

Risk, in the design context, is the threat or probability that an action or event will adversely affect a system to achieve its objectives. Risks are very common in a design. If a design does not satisfy the goals or the requirements of a system, then a risk is arisen. A risk could be in non-functional requirement such as a lack of performance or a security breach, it could be that the functional requirements are not satisfied, or that the quality of the system is not met.

When designing a system, designers must be cognizant of the potential shortfalls of a design. We suggest that risks can be of two general sorts. Firstly, a designer is not aware of the behaviour of the design components and if the design components would satisfy the requirements or not. This risk is due to the lack of knowledge about how a design would behave. We call this Outcome Certainty Risk. In order to mitigate this risk, a designer would need to drill the design into further details until such risk is well understood and minimised.

Secondly, there is a risk that a design and implementation team is not capable of implementing a system. This can be due to the lack of experience (i.e. knowledge) of the business domain, the technology being used, the skill set of the team and other factors. It is important that we recognise such a risk and deal with it appropriately [13].

There are three mitigation strategies. Firstly, simplify the design solution or use other appropriate technologies that the development team is capable to deal with. Secondly, hire experienced development team that can demonstrate the ability to deal with the specific issues. Thirdly, compromise the requirements and design concerns to reduce its complexity.

Generally, just be aware of the potential risks that stop your design from working properly. Ask yourself these questions when you consider each design decision:

- Are there any hidden assumptions behind this decision? What are they?
- Are there any unknowns in the design that could adversely affect your design?
- What is the certainty that your decision and the resulting design is sound?
- Does this decision create new risks for the rest of the solution?

3.6. Deductive Reasoning

Deductive reasoning evaluates the deductive arguments. An argument is said to be deductive when the truth of the conclusion is purported to follow a logical consequence of the premises and (consequently) its corresponding conditional is a necessary truth. Deductive arguments are said to be valid or invalid, never true or false [14]. Let us use an example to illustrate deductive reasoning. If a hard disk is full, the average time to seek and retrieve the data in it takes longer time and the performance of the disk would eventually degrade. If you insert a lot of records into a table, the hard disk becomes full. Putting the two arguments together, one can deduce that inserting a lot of records into the database can degrade the performance of a hard disk.

This kind of reasoning seems like common sense except that designers need to note two things to employ deductive reasoning effectively in design. Firstly, the fundamental argument or proposition must be factual and valid. Designers must be careful not to assert arguments that are without proper factual support and based on the “of course it is true” assertion. If the

premises are false, so will be the results of the deductive reasoning. For instance, if we assert that "all software has bugs" and "print Hello world program is a piece of software", then the deductive reasoning result is "print Hello program has bugs". This is obviously untrue.

Secondly, designers must not leave out key assumptions or facts in such arguments. For instance with the last example, if there is an archival process which removes data record periodically, then the performance of the hard disk can be maintained.

Deductive reasoning is a general technique and it should be used in combination with the other reasoning techniques such as risking analysis, constraint analysis and option analysis etc. The example about the performance of a disk highlights the possible constraint on disk performance. Designers should be trained to think critically, logically and factually, and learning such skills from resources such as [15].

3.7. Inductive Reasoning

With inductive reasoning, a designer observes certain events and formulates a hypothesis from the observations of those events. This is a kind of generalisations from observations. Often designers, or human beings in general, form opinions from such inductive reasoning. However, such hypothesis may or may not be true. For instance, a designer may observe that a number of commercially released software in the operating system and database platforms are buggy, and reason that all newly released software is buggy. The truth of this hypothesis depends on the extent of the generalisation. For instance, if the observations is about one particular software vendor, and this vendor has a poor record of software quality assurance, then this generalisation is likely to be true for this specific software vendor. However, there are software vendors with vigorous quality assurance process and therefore this hypothesis is unlikely to be applicable.

Hypothesis that is formed from inductive reasoning can be context dependent. That is, the context of the observations influences the applicability of the generalisation. For instance, an embedded system designer is asked to design a database application system. S/he is accustomed to minimising the memory footprint of an executable because of the embedded system environment, and s/he puts on a lot of efforts into minimising the memory footprint of a database application. His/her reasoning is that all software needs to have a minimal memory footprint. This, however, is out of context for a database system since all SQL query cursors are managed by the system anyway. His/her design effort is not productive because the argument that is used is outside of the relevant context of what is being argued. So when we apply design generalisations, i.e. apply inductive reasoning, we must be careful with the extent and the context of our arguments.

4. Appendix A - Record Your Reasoning Technique

As an exercise to get familiar with using and assessing design reasoning techniques, we have prepared a sample table which designers and students can use to log the use of design reasoning techniques. It can help designers and students assess if the techniques help them identify design gaps and if they improve their decision making.

Decision	What reasoning technique did you use in your design?	Describe the design issues.	How does the reasoning techniques work for you?
Example Decision name – new decision or modify previous decision	Example 1.We used problem structuring to identify an unknown performance requirement in database access	Example Missing performance consideration in creating an index to address the performance of INSERT in XXX table,	Example 1. This tactic help us to XXXXX Or 2. We tried to use XXX tactic and found that XXXX

5. References

[1] R. Bartk, "Modelling Soft Constraints: A Survey," Charles University in Prague , Faculty of Mathematics and Physics, Praha 2002.

[2] M. van den Berg, A. Tang, and R. Farenhorst, "A Constraint-Oriented Approach to Software Architecture Design," in *n Proceedings of the Quality Software International Conference (QSIC 2009)*, 2009.

[3] J. S. Evans, "In two minds: dual-process accounts of reasoning," *Trends in Cognitive Sciences,* vol. 7 (10), pp. 454-459, 2003.

[4] J. Evans, "Heuristic and analytic processes in reasoning," *British Journal of Psychology,* vol. 75 (pp. 451-468, 1984.

[5] A. Tversky and D. Kahneman, "Rational Choice and the Framing of Decisions," *The Journal of Business,* vol. 59 (4 Part 2), pp. S251-S278, 1986.

[6] C. Zannier, M. Chiasson, and F. Maurer, "A model of design decision making based on empirical results of interviews with software designers," *Information and Software Technology,* vol. 49 (6), pp. 637-653, 2007.

[7] A. Tang, M. H. Tran, J. Han, and H. van Vliet, "Design Reasoning Improves Software Design Quality," in *Proceedings of the Quality of Software-Architectures (QoSA 2008)*, 2008.

[8] J. Carroll and E. Johnson, *Decision research : a field guide* Sage, 1990.

[9] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '98)*, 1998, pp. 68-78.

[10] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston: Addison Wesley, 2003.

[11] J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software,* vol. 22 (2), pp. 19-27, 2005.

[12] A. Tang, J. Han, and R. Vasa, "Software Architecture Design Reasoning: A Case for Improved Methodology Support," *IEEE Software,* vol. Mar/Apr 2009 (pp. 43-49, 2009.

[13] A. Tang and J. Han, "Architecture Rationalization: a Methodology for Architecture Verifiability, Traceability and Completeness," in *12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems ECBS 2005*, U.S.A., 2005, pp. 135-144.

[14] Wikipedia, "Deductive reasoning."

[15] D. F. Halpern, *Thought and Knowledge: An Introduction to Critical Thinking*: Lawrence Erlbaum Associates 2002.